OS_HW2 1/6

#3. Synchronization by Semaphore

#1. Race Condition 의 문제를 Semaphore 를 이용하여 해결하여라.

다음은 #1 과제 당시 작성했던 프로그램 일부입니다. 2개 이상의 스레드 생성 시 정상적으로 작동하지 않았습니다.

```
void *_ simpleThread__(void *data) {
    int id;
    char tmp[10];
    int memoryValue;
    int i = 0;
    id = *((int*)data);
    for (i = 0; i < 100; i++) {
     sleep(1);
     if (i < 100) fseek(sharedMemory, -3, SEEK_END);</pre>
     else if (i < 1000) fseek(sharedMemory, -4, SEEK_END);
     else errHandling("value overflow");
     fscanf(sharedMemory, "%d", &memoryValue);
     fprintf(sharedMemory, "%d\n", memoryValue+1);
     printf("[%d][Thread ID : %d] get %d, put %d \n",
                ++runTimes, id, memoryValue, memoryValue+1);
    return 0;
```

[이 당시 원하는대로 값이 나오지 않았습니다.]

이 당시 각 스레드끼리 동기화가 되어있지 않아 하나의 공유자원에 서로 접근하여 원하는 결과를 내지 못하는 상황이 발생하였습니다. 세마포어를 두어 하나의 공유자원에 동시에 하나의 스레드 이상 접근하지 못하게 수정할 필요가 있었고, 따라서 다음과 같이 프로그램을 수정하였습니다.

다음은 수정된 코드의 전문입니다.

OS_HW2 2/6

```
// Filename : OS_HW3.c
// Usage : ./filename <number(s) of thread(s) : 1 to 3>
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <pthread.h>
#include <unistd.h>
                           // POSIX operating system : for Distribute Computing
#include "semaphore.h"
#define ONE THREAD '1'
#define TWO_THREAD '2'
#define THR THREAD '3'
#define VAL_LENGTH 4
FILE* sharedMemory;
static int runTimes;
void errHandling(char* msg) {
       fputs(msg, stderr);
       fputc('\n', stderr);
       exit(1);
}
void initMemory() {
       sharedMemory = fopen("./physicalMemory.txt", "w+");
       fprintf(sharedMemory, "[Thread ID : X] 0\n");
       runTimes = 0;
       return;
}
struct params {
       int pid;
       key_t skey;
};
void *__simpleThread__(void *data) {
       int pid;
       int memoryValue;
       int i = 0;
       int semid;
       struct params *param = data;
       pid = param->pid;
       if (( semid = initsem(param->skey, 1) ) < 0) errHandling ("init Semid Err.");
```

OS_HW2 3/6

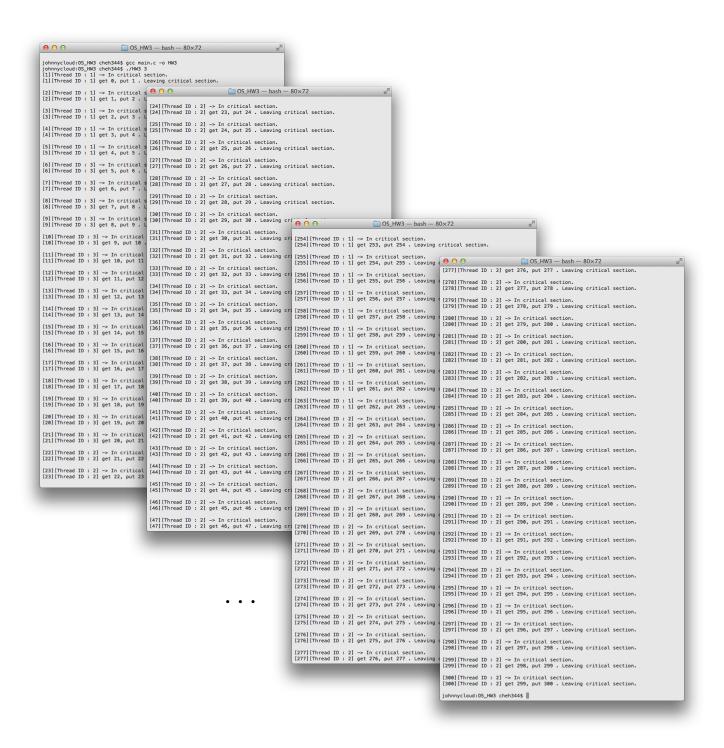
```
for (i = 0; i < 100; i++) {
             /*********** critical section begin *********/
             p(semid);
             printf("[%d][Thread ID : %d] -> In critical section. \n",++runTimes, pid);
             usleep(50000); // 0.05 second waits.
             if (runTimes < 100) fseek(sharedMemory, -3, SEEK END);
             else if (runTimes < 1000) fseek(sharedMemory, -4, SEEK_END);
             else errHandling("value overflow"); // it will not happen in this code.
             fscanf(sharedMemory, "%d", &memoryValue);
             fprintf(sharedMemory, "[Thread ID : %d] %d\n", pid, memoryValue+1);
             printf("[%d][Thread ID: %d] get %d, put %d. Leaving critical section.\n\n",
                      runTimes, pid, memoryValue, memoryValue+1);
             v(semid);
              /************ critical section end **********/
      return 0;
}
void threadRace(char thrNum) {
      pthread_t p_thread[3];
      int thrID, status;
       /*
      int a = 1;
      int b = 2;
      int c = 3;
      struct params a;
      struct params b;
      struct params c;
      a.pid = 1;
      b.pid = 2;
      c.pid = 3;
      a.skey = b.skey = c.skey = 0x200; // [in sample code]
      if (thrNum == ONE THREAD) {
             thrID = pthread_create(&p_thread[0], NULL, __simpleThread__, (void*)&a);
             if (thrID<0) errHandling("Err: thread create failed");
             // wait for thread end
```

OS_HW2 4/6

```
pthread_join(p_thread[0], (void**)&status);
      }
      else if (thrNum == TWO_THREAD){
              thrID = pthread_create(&p_thread[0], NULL, __simpleThread__, (void*)&a);
             if (thrID<0) errHandling("Err: Thread[id:1] create failed");
             thrID = pthread_create(&p_thread[1], NULL, __simpleThread__, (void*)&b);
             if (thrID<0) errHandling("Err: Thread[id:2] create failed");
             // wait for thread end
              pthread join(p thread[0], (void**)&status);
              pthread_join(p_thread[1], (void**)&status);
      else if (thrNum == THR_THREAD){
              thrID = pthread_create(&p_thread[0], NULL, __simpleThread__, (void*)&a);
             if (thrID<0) errHandling("Err: Thread[id:1] create failed");
              thrID = pthread_create(&p_thread[1], NULL, __simpleThread__, (void*)&b);
             if (thrID<0) errHandling("Err: Thread[id:2] create failed");
             thrID = pthread create(&p thread[2], NULL, simpleThread , (void*)&c);
             if (thrID<0) errHandling("Err: Thread[id:3] create failed");
             // wait for thread end
              pthread_join(p_thread[0], (void**)&status);
              pthread join(p thread[1], (void**)&status);
             pthread_join(p_thread[2], (void**)&status);
      }
      else errHandling("wrong Thread num.");
}
int main(int argc, char **argv){
      if (argc != 2) errHandling("Usage : ./filename < number(s) of thread(s) : 1 to 3> ");
      initMemory();
      switch (*argv[1]) {
              case ONE THREAD:
                    threadRace(ONE_THREAD);
                    break;
              case TWO THREAD:
                    threadRace(TWO THREAD);
                    break;
              case THR_THREAD:
                    threadRace(THR_THREAD);
                    break;
              default:
                    errHandling("wrong ipt");
      }
      return 0;
}
```

OS HW2 5/6

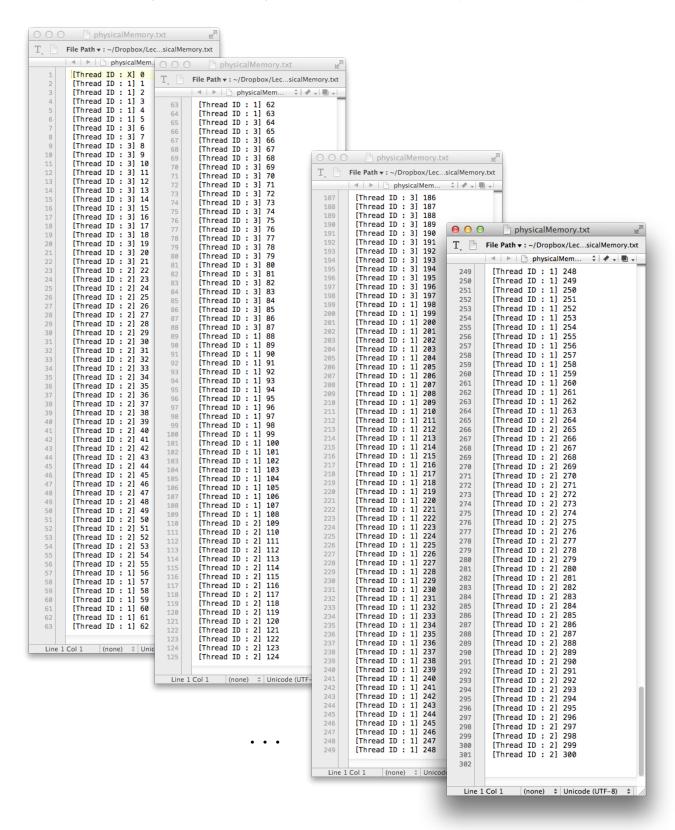
다음은 3개의 스레드를 실행하였을 때의 터미널 출력결과입니다. 프로그램에 세마포어를 추가하여 수정함으로 써, 상호배제가 정상적으로 일어남과 동시에 결과 또한 정상적으로 출력됨을 확인할 수 있었습니다.



[원하는 결과가 나왔습니다.]

OS_HW2 6/6

다음은 공유자원에 출력된 결과입니다. 출력된 숫자 앞에 어떤 스레드가 추가했는지 표시를 남겨두었습니다.



[출력 결과에 이상은 없었습니다.]